

GWT From Scratch – Day 3

The Listeners

The aim of today's session is to explain a bit about Listeners.

You will

- Have a look at a number of ways of implementing listeners using the ClickListener as an example
- Have a quick look at how the compiler sees your program
- Use a FocusListener
- Use a MouseListener
- Learn how to roll your own EventListener

Questions?

My ideal is to have no questions at all because everything is perfectly clear, but if you have any questions, then please contact me. I would like to consider the course 'complete' within the limits I have set for it. In other words, I am here to supplement the course if, in any way, it is not well enough explained to get everyone through it. I intend to use the feedback to improve the course and reduce the questions. So any questions are very welcome.

Any Problems

rx01-day3@examples.roughian.com

© Ian Bambury 2008

Listeners

GWT is an event-driven language. What does that mean? Well it is the next big thing after the command line. With command line and programming you type in what you want to happen and press enter. That's it.

Event-driven programming is (pretty obviously) driven by events. Now that can mean a key-press, or a mouse-move, or a mouse-click. Those are the things that you usually think of. Then you might think about a control getting or losing focus, or maybe the text in TextBox changes.

But there are plenty of other things you can use. It might be a touch on the screen, or a change in temperature or something breaking a light beam, or a number in a database reaching its limit. If you can detect it anywhere in a computer or its peripherals, then it's an event you can program for.

Common Listeners

ChangeListener	Text has changed
ClickListener	Something has been clicked
EventListener	A catch-all for when ANYthing happens
FocusListener	Something has got or lost focus
HistoryListener	The user/program has navigated history.
KeyboardListener	Keypress, or down or up
LoadListener	Something has loaded
MouseListener	The mouse has done something in your widget(!)
MouseWheelListener	The mouse wheel has moved
PopupListener	A popup has popped up or down
ScrollListener	Scrolling has occurred
TableListener	A cell has been clicked
TreeListener	An item has been selected, opened, or closed
WindowCloseListener	Window closing
WindowResizeListener	The window is a different size

ClickListener

We had a quick look at this yesterday. It's a nice, simple listener that only reacts to one event: a click. We attached it to a button (or maybe we attached a button to it, I'm never sure - they became linked, anyhow).

We had a button at the class level, and a listener at the class level, and in the `onModuleLoad()` method, we joined the two of them up. If you take out all the irrelevant stuff, you end up with something like this:

```
public class Main implements EntryPoint
{
    Button button = new Button("Transfer");

    public void onModuleLoad()
    {
        button.addClickListener(listener);
    }

    ClickListener listener = new ClickListener()
    {
        public void onClick(Widget sender)
        {
        }
    };
}
```

We could have done this:

```
public class Main implements EntryPoint
{
    Button button;

    public void onModuleLoad()
    {
        button = new Button("Transfer", listener);
    }

    ClickListener listener = new ClickListener()
    {
        public void onClick(Widget sender)
        {
        }
    };
}
```

Here we have declared the button at the class level, but not instantiated it - i.e. we have a name for it (`button` - lower case), and we know what this named-thing can hold (a `Button` object - object has an initial capital) but it isn't actually anything yet. It's like a plot of land with planning permission for a certain type of building.

In the `onModuleLoad()` we instantiate it (i.e. we now have a real button in it, and while we create it, we tell it the text to put in it, and also the listener to associate it with).

Next, we create the listener.

Now I say 'next' but it is only next as you read down the code. In ancient times, compilers would be 'single pass' - they would read through the file once, and the

above code wouldn't work because you can't assign a listener before you have created it.

That is still true in multi-pass compilers, but first they go through all the class-level stuff and create that, and then they go through the methods, creating as they see fit. So this next example wouldn't work.

```
public class Main implements EntryPoint
{
    Button button = new Button("Transfer", listener);

    ClickListener listener = new ClickListener()
    {
        public void onClick(Widget sender)
        {
        }
    };
}
```

But this would work:

```
public class Main implements EntryPoint
{
    ClickListener listener = new ClickListener()
    {
        public void onClick(Widget sender)
        {
        }
    };
    Button button = new Button("Transfer", listener);
}
```

because the listener is created and only then is it assigned.

Why do you need to know all this? Because program layout is important and with a two-pass compiler can be arranged top-down. You can start with the main routine so anyone looking at the code can get an overview. If they need more detailed information, they can keep reading. In the old days, it was literally bottom-up - you started reading the last pages first and worked your way up.

Yeah, but who cares, and why are you banging on about it?

It matters. It will matter to you when you come back to one of your own programs after six months and you can't remember a damn thing about it. Someone else might just as well have written it.

There are quite a number of ways you can write listeners into your code, and the best is the simplest-looking one you can find. Simple and you can sum up what they do immediately.

So here we go then with a number of ways of doing it.

My Favourite

You turn the class itself into a listener and it listens to something inside itself. Here's a little example that does nothing useful.

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint, ClickListener
{
    public void onModuleLoad()
    {
        Button button = new Button("Click Me");
        RootPanel.get("slot").add(button);
        button.addClickListener(this);
    }

    public void onClick(Widget sender)
    {
        Window.alert("I Was Clicked!");
    }
}
```

So what is happening here?

Notice 'ClickListener'. This means that it promises that it will do everything that ClickListener should do. In the case of a ClickListener, we need an onClick() method.

So instead of saying that we will create a listener especially for this button, what we are saying is that we'll make this actual class a listener.

So we create a new button, we get our slot and add the button to it, and then we add a ClickListener. Notice the use of 'this' which is a Java class's way of saying me, myself, and I.

You might also have noticed the Window.alert(). The Window object allows you to do a few things related to the browser window. In this case the program is popping up a message box. This is a good way to find out if a method is actually getting called or to find out the value of something when you are debugging.

As you can see, this is a very compact way of dealing with the problem of adding a listener and making the code readable. The only thing you might want to do is to make sure that you keep the onClick() method close to the addClickListener().

"That's all very well," I hear you say. "But what happens if I've got more than one button? Or if I want to react to the user clicking something else as well?"

A very good question.

Anonymous Listeners

But what you do if you're not creating your own class? What if you're just in an ordinary method and sticking ordinary buttons on the screen?

Well for that, the simplest way is to have an anonymous listener created at the same time as you create the button.

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint
{
    public void onModuleLoad()
    {
        Button button = new Button("Click Me", new
ClickListener()
        {
            public void onClick(Widget sender)
            {
                Window.alert("I Was Clicked!");
            }
        });
        RootPanel.get().add(button);
    }
}
```

This is a ClickListener just for this one button. Because you haven't declared it separately, you can't refer to it because it doesn't have a name (i.e. it is 'anonymous').

You would use this in a situation where you have just one object (for example, as in this case, a button) which requires a dedicated listener.

You wouldn't use this method if there was ever going to be more than one instance of the same ClickListener. For example, if you had a number of rows of items in a shopping cart on the screen, and for each line there was a button which allow the user to remove the line from the shopping cart, you would not use this method. You could use it, and it would work, but it would create a ClickListener for every button which isn't strictly necessary, and would use more memory.

So how do we have one listener for a number of buttons (or any other widget), and how does the ClickListener know which button was pressed?

Reusable ClickListener

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint
{
    public void onModuleLoad()
    {
        Button button1 = new Button("Click Me", listener);
        Button button2 = new Button("No, Click Me",
listener);
        RootPanel.get().add(button1);
        RootPanel.get().add(button2);
    }
    ClickListener listener = new ClickListener()
    {
        public void onClick(Widget sender)
        {
            Window.alert("I Was Clicked!");
        }
    };
}
```

Here we have created a separate ClickListener and assigned it to two different buttons. The problem is, of course, how do we know which button was clicked?

The 'sender' which is passed to the onClick() routine is whatever was clicked. It is passed as a Widget which is the base class that all GWT widgets are, erm, based. If you're not from a Java background (or any other object-oriented language) then a very quick analogy is that although you and I know that we are being passed a rabbit, what we are being passed has been labelled as `type = animal` and we To relabel it as `type = rabbit` before we can use any of its 'rabbitty' features. We do that by 'casting' the Widget to a Button.

```
ClickListener listener = new ClickListener()
{
    public void onClick(Widget sender)
    {
        Button button = (Button) sender;
        Window.alert("My text is '" + button.getText() +
    "'");
    }
};
```

In the highlighted line, we are creating an 'object-container' for a button and we are squishing 'sender' into it as a button. Now this will work okay in the above program,

but if you assigned the ClickListener to a Label, for example, then you would get an error if the user clicked on the label. The only reason it will continue working is if you remember only to send buttons to it.

So in this example, we have a single ClickListener dealing with multiple objects. If you moved the ClickListener inside the onModuleLoad() routine, then you would have to declare it before you declared button1 and button2.

Other Events And Listeners

The FocusListener

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.FocusListener;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint, FocusListener
{
    public void onModuleLoad()
    {
        TextBox textBox1 = new TextBox();
        RootPanel.get().add(textBox1);
        textBox1.addFocusListener(this);

        TextBox textBox2 = new TextBox();
        RootPanel.get().add(textBox2);
        textBox2.addFocusListener(this);
    }

    public void onFocus(Widget sender)
    {
        TextBox textBox = (TextBox) sender;
        textBox.setText("Got Focus");
    }

    public void onLostFocus(Widget sender)
    {
        TextBox textBox = (TextBox) sender;
        textBox.setText("Lost Focus");
    }
}
```

In this example we are making Main class the FocusListener. The two required methods for a FocusListener are onFocus and onLostFocus. Again we are 'casting' the sender, but this time to a TextBox so that we can use its setText() method to set the text in the box just to prove that it all works.

The MouseListener

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.FocusListener;
import com.google.gwt.user.client.ui.HasAlignment;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.MouseListener;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint, MouseListener
{
    public void onModuleLoad()
    {
        Label label = new Label();
        RootPanel.get().add(label);
        label.addMouseListener(this);

        label.setHorizontalAlignment(HasAlignment.ALIGN_CENTER);
        label.setText("Nothing Has Happened Yet");
    }

    public void onMouseDown(Widget sender, int x, int y)
    {
        ((Label) sender).setText("onMouseDown");
    }

    public void onMouseEnter(Widget sender)
    {
        ((Label) sender).setText("onMouseEnter");
    }

    public void onMouseLeave(Widget sender)
    {
        ((Label) sender).setText("onMouseLeave");
    }

    public void onMouseMove(Widget sender, int x, int y)
    {
        ((Label) sender).setText("onMouseMove");
    }

    public void onMouseUp(Widget sender, int x, int y)
    {
        ((Label) sender).setText("onMouseUp");
    }
}
```

In this example we are catching mouse events. Our Main class implements MouseListener and therefore it also has to implement onMouseDown, onMouseEnter, onMouseLeave, onMouseMove, and onMouseUp.

In the `onModuleLoad` method, have a look at the `setHorizontalAlignment` which is how you set the horizontal alignment of anything that can have horizontal alignment.

I've highlighted the `int x` and `int y` parameters. This is, of course, passing you the pixel points of the mouse's position at the point when they user presses the mouse button down. It's

Also note the `((Label) sender).setText("onMouseEnter");` which is a quick way to set the text without having the extra line we had in the previous example where we cast the sender to a `TextBox` in one line and then set the text in the next line. Here we are cutting out the middleman. This is only useful if you are only going to do one thing with the sending object because you don't want to have to do that every time. Personally, if the line got any more complicated than and split it up for readability reasons, but if you are a C-programmer then you will properly try to get the whole program on one line.

Other Listeners

The other listeners fall into two categories: those that are very similar to the three that I have described (like the `ChangeListener`), and those which are quite specialised (like the `HistoryListener`). There is little point in going through the ones which are very similar because you won't really learn anything new. And more specialised ones like the `HistoryListener` require a better understanding of GWT than you have at the moment if all you have done so far is this course.

There is one that I will cover, however. Instead of being a specialised listener, it is a role-your-own event listener which you can use to access events that aren't covered by the other event listeners or if you need to respond to an unusual set of events.

The EventListener

The `EventListener` as I have just said allows you to listen or pretty much any event that there is within a browser. Most of these are covered by other event listeners, but we'll be looking at other things too.

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.Event;
import com.google.gwt.user.client.EventListener;
import com.google.gwt.user.client.ui.HasAlignment;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.MouseWheelVelocity;
import com.google.gwt.user.client.ui.RootPanel;

public class Main implements EntryPoint, EventListener
{
    int count = 0;
    Label label = new Label("Hello");

    public void onModuleLoad()
```

```

    {
        label.sinkEvents(Event.ONMOUSEWHEEL);
        RootPanel.get("slot").add(label);

        DOM.setEventListener(label.getElement(), this);

        label.setWidth("100px");

        label.setHorizontalAlignment(HasAlignment.ALIGN_CENTER);

        DOM.setStyleAttribute(label.getElement(),
                               "border", "1px dotted red");
        DOM.setStyleAttribute(label.getElement(),
                               "padding", "20px");
        DOM.setStyleAttribute(label.getElement(),
                               "margin", "20px auto");
    }

    public void onBrowserEvent(Event event)
    {
        switch (DOM.eventGetType(event))
        {
            case Event.ONMOUSEWHEEL:
                MouseWheelVelocity velocity =
                    new MouseWheelVelocity(event);
                count += velocity.getDeltaY();
                label.setText("Count = " + count);
                DOM.eventPreventDefault(event);
                break;
        }
    }
}

```

The idea of this little program is that we're going to create a label which counts the number of scrolls they user makes when the mouse pointer is over that label. Totally pointless.

We declare the Main class as an `EventListener`, but you could have some other class, or you could just have a straight variable declaration.

You have to 'sink' events that you want to monitor. The browser would grind to a halt if it had to monitor every event for every object. In this case we want to monitor the movement of the mouse wheel.

The application will now know it has to listen to mouse-wheel events for label, but it doesn't yet know what to do when he gets one.

The DOM object is GWT's way of letting you get at the Document Object Model and here we use the `setEventListener` method. We pass it the label's element (i.e. the little chunk of the Document Object that is our label), and the object which is going to handle the event (in our example program this is 'this' - the Main class. The `EventListener` interface tells the system that there must be an `onBrowserEvent` method.

Carrying on down the program, `label.setWidth("100px");` as you might guess sets the width of the label to 100 pixels wide. This will accept anything that is a valid expression for width in CSS.

Back to the DOM for the next line.

```
DOM.setStyleAttribute(label.getElement(), "border",  
    "1px dotted red");
```

You can of course use CSS to style your widgets

Just a side note - I'm using Dragon NaturallySpeaking to dictate this and when I said 'style your widgets' it typed 'start all idiots' - the microphone wasn't placed correctly.

If you see `setStyleAttribute` method then you have to recompile program in order to change something fairly simple, but on the other hand, if it's something that you don't want designers to play around with, then it's a good choice :-)

You pass it the element of the object you want to style, and then the attribute that you want to style in the camelCase, then the value for it.

On to the actual handler, then.

```
public void onBrowserEvent(Event event)  
{  
    switch (DOM.eventGetType(event))  
    {  
        case Event.ONMOUSEWHEEL:  
            MouseWheelVelocity velocity =  
                new MouseWheelVelocity(event);  
            count += velocity.getDeltaY();  
            label.setText("Count = " + count);  
            DOM.eventPreventDefault(event);  
            break;  
    }  
}
```

We use a 'switch' statement to choose which event we are going to handle. A switch statement is the same as a 'Select Case'.

In the case of an `Event.ONMOUSEWHEEL`: occurring, the next bit of code will be run. `MouseWheelVelocity` is an object that can tell you all about what's been happening with your users mouse wheel. So we get it for this event. `getDeltaY` get you the change in the amount of scroll so we had it to the count variable (it can be positive or negative). Then we set the text of the label.

The `eventPreventDefault` line is to ensure that nothing else will get the event. It is like a `cancelBubble` if you know what that is. Without this line if the page could scroll in the browser, then it would.

Q & A

Dear Ian,

How do I know what the listener applicable for each widget is? And what are the methods available in the each widget? Can you send me each widget's listener and listener's methods in table format?

Thanks.
Saravanan.

Hi Saravanan,

I don't have a prepared list, but there are two references where you can look them up - on my site, of course, you can look up a widget to see what listeners are available, and you can look up a listener to see an example - every example has all the methods in a working example.

A more technical reference (note this is for 1.4) is available from Google here:

<http://google-web-toolkit.googlecode.com/svn/javadoc/1.4/index.html?com/google/gwt/user/client/ui/>

You can look up, say, `MouseWheelListener` and get a complete reference including all the events with parameters and descriptions. To look the other way (to find out what has these listeners) look for the Sources... For example `SourcesMouseWheelEvents` shows you every class that implements the interface (therefore, every class where the `MouseEventsListener` will be available)

In practice, in Eclipse, to find out what listeners are available for a widget, I would just type the name of the widget plus `'.add'` - the autocomplete shows you all the methods starting with `'add'` so all the `addWhateverListener` methods will be shown.

To create a listener in your code, I never remember what all the methods are, or type them in. There's an easier way:

Number 2 Time-Saving Tip in Eclipse.

Use `Control+1` to correct errors. More than that, just type enough code so Eclipse knows what you are trying to do, and then use `Control+1` to complete it.

The number 1 time-saver is, of course, to use autocomplete (don't forget to press `Control + SpaceBar` if autocomplete has gone missing)

For a mouse listener, for example, type:

```
MouseListener listener = new MouseListener(){};
```

in Eclipse. Make sure that your cursor is on that line (the line will be flagged with an error because the methods aren't there)

Press Control+1 and you will be offered the chance to 'Add unimplemented methods'. If you choose it, it will add everything which is missing - cool, huh? No typing, nothing forgotten. No confusion as to what it really wants. No looking up what the parameters should be.

I use it a lot to save me typing stuff in or looking things up.

Any other questions, let me know.

Ian

Tomorrow

Is all about the layout of your screen using, mainly, Panels of some sort - and the ways of styling the things on your screen.